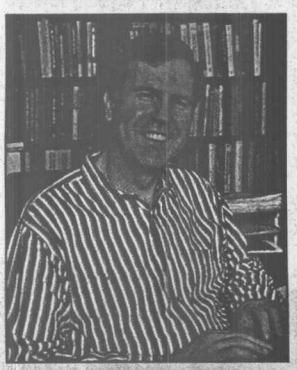


DAVID BRUBAKER,
CONTRIBUTING EDITOR

Automated fuzzy-rule-based design

If the union of fuzzy-rule-based and neural-network technologies described in this issue were a marriage between two people, it would certainly be dysfunctional, with one partner (the neural net) always telling the other (the fuzzy rule base) what to do. As it is, this particular marriage of technologies functions quite well.

The use of a neural network to design the components of a fuzzy-rule-based system is currently the most visible combination of neural-net and fuzzy-rule-based technologies. You use this technique when a legitimate domain expert is not available and when a rule base is the desired structure. Of course, the training data must also be reliable and complete. The data consists of samples of the response function the fuzzy-rule-based system is required to generate. This column discusses a general approach to a neural network that is used to design a fuzzy rule base without delving into the details of the individual systems.

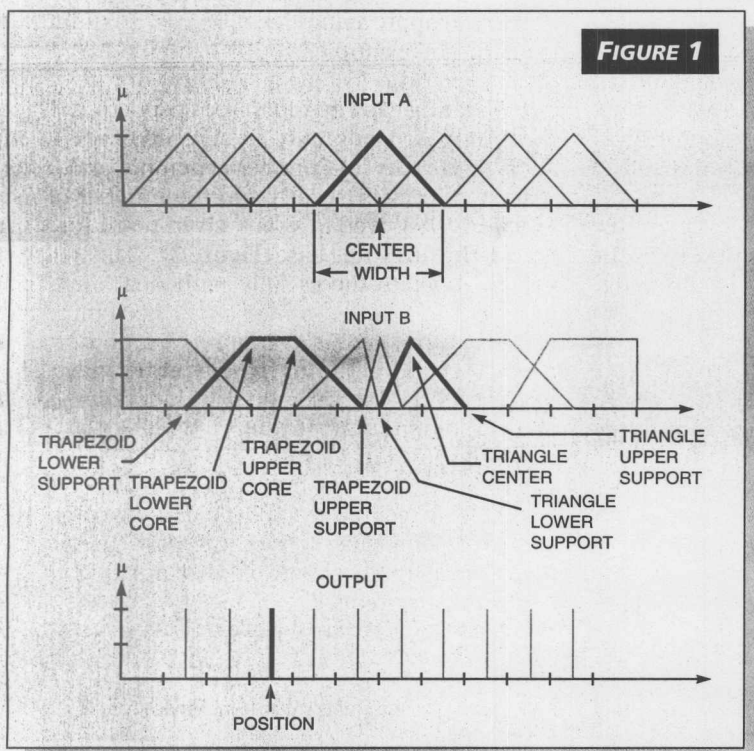
You can use a neural network to define both rules and membership functions of the fuzzy-rule-based system. Most often, however, the system designer defines rules to create a generally acceptable system with a generic set of membership functions, and the neural network then modifies the membership-function positions and shapes to achieve improved performance.

As an example, consider the membership functions of a two-input, single-output fuzzy-rule-based system (**Figure 1**). Input A's membership functions consist of five isosceles triangles, and Input B's membership functions consist of three acute triangles and four trapezoids. Ten output singletons make up the single Output. All membership functions have unity maximums.

A single parameter is all that is necessary to completely describe a singleton—its position on the axis. Two parameters completely describe each isosceles triangle: the position of the triangle's center on the axis and its width. You can completely describe each nonisoceles triangle using three parameters: positions on the axis of the lower support boundary, where μ first becomes nonzero; of the maximum, where $\mu=1$; and of the upper support boundary, where μ returns to zero. Finally, four parameters completely describe a trapezoid: positions on the axis of the lower support boundary, where μ first becomes nonzero; of its lower core boundary, where μ first becomes unity; of its upper core boundary, where μ first drops away from unity; and of its upper support boundary, where μ returns to zero.

A system designer can use a neural network to manipulate these parameters and, thereby, to specify membership functions in a number of ways. **Figure 2** shows one of the most straightforward methods, which, although it has problems handling sparse data sets and large fuzzy rule bases, works

FIGURE 1



Input A's membership functions are isosceles triangles, and Input B's membership functions are either triangles or trapezoids. Isosceles triangles are fully described by their lower and upper support boundaries and the center of the triangle. Other triangles and trapezoids are fully described by their lower and upper support boundaries and their lower and upper core boundaries. Singletons in the output are described by position only. A neural net can optimize these parameters and the membership functions for a given set of training data.

PHOTOGRAPHY / LIANE ENKELIS

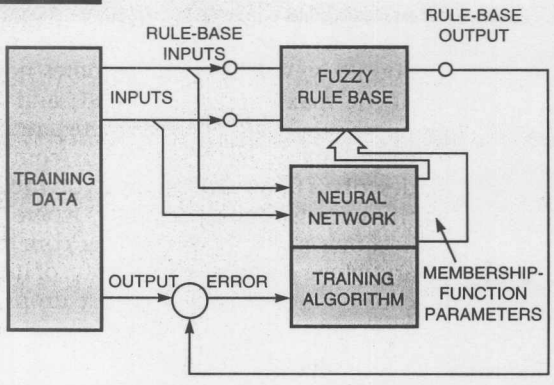
well with simpler systems. I present it here because it is easy to understand.

The training process involves both the neural network and the fuzzy rule base. You apply training data inputs—data that the fuzzy rule base sees during its normal operation—to both rule base and neural net, and the neural net responds with modified versions of the fuzzy-membership functions. Using these, the fuzzy rule base generates an actual output, which is compared with the desired output contained in the training data. The training algorithm changes the neural-net weights until the difference between the actual and desired outputs for the first set of training data is minimized. Each additional set of training data is iteratively applied until the final membership function parameters and, therefore, the membership-function shapes, converge to their final values. With membership functions defined in this manner, operation of the fuzzy rule base closely mimics the operation represented by the training data.

Care must be taken in two significant areas, and this care applies to all applications of neural networks. First, data used to train the neural net must be representative, accurate, and complete. A correctly constructed neural network accurately mimics the response function presented by its training data, and, if the training data is faulty, the neural net operation is faulty. Unrepresentative or inaccurate data results in a neural net that generates an inaccurate response function. When incomplete data is used, the generated response function is accurate where the data is good, but it is difficult to anticipate where the data is not good.

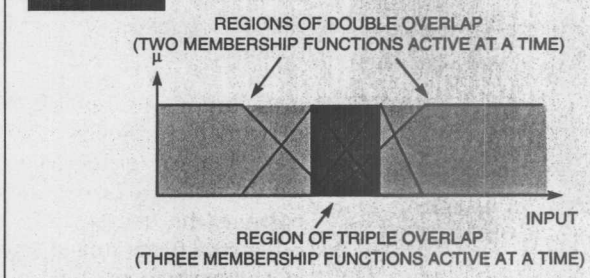
The second area of concern is the need to structure the neural net appropriately for the size and complexity of the undertaken task. Being able to do this requires knowledge and experience working with neural networks.

FIGURE 2



To train a neural net, apply input data to both the fuzzy rule base and the neural net, allowing the neural net to modify the rule-base membership functions to minimize the difference between the actual output of the rule base and the desired output contained in the training data.

FIGURE 3



Most fuzzy systems have, at most, two membership functions that are active for any given input value. For additional capability, three overlapping membership functions may sometimes be needed, but you rarely need more. Adding a three-active-membership function constraint to the neural network's outputs may be necessary to speed the training process and retain the intuitive nature of the resulting fuzzy-membership functions.

You can apply a number of "tricks" to simplify the design process. For example, you can divide the fuzzy system's input space into its four quadrants with predefined boundaries, treating each quadrant independently. The neural net then designs each quadrant, not the entire rule base. This approach results in a simpler neural net and faster training.

Another trick is to constrain the number of membership functions that overlap at any given point for a single input within the fuzzy rule base (**Figure 3**). Many fuzzy-rule-based systems operate successfully with dual membership-function overlap, and few systems ever need more than a triple membership-function overlap. If you do not constrain the neural network's output, it may either never converge or may design a set of membership functions that are unnecessarily complex.

A claim I often make in this column is that there is no magic in fuzzy logic; that it should be considered a solid engineering technique applicable in the solutions to a wide variety of problems. I hope the past three columns have demonstrated that there is also nothing magical about neuro-fuzzy systems. EDN

VOTE

Please use the Information Retrieval Service card to rate this article (circle one):

High Interest	Medium Interest	Low Interest
578	579	580

David Brubaker is a consultant in fuzzy-system design. You can reach him at Huntington Advanced Technology, 883 Santa Cruz Ave, Suite 31, Menlo Park, CA 94025-4608 or on the Internet at: brubaker@cup.portal.com.